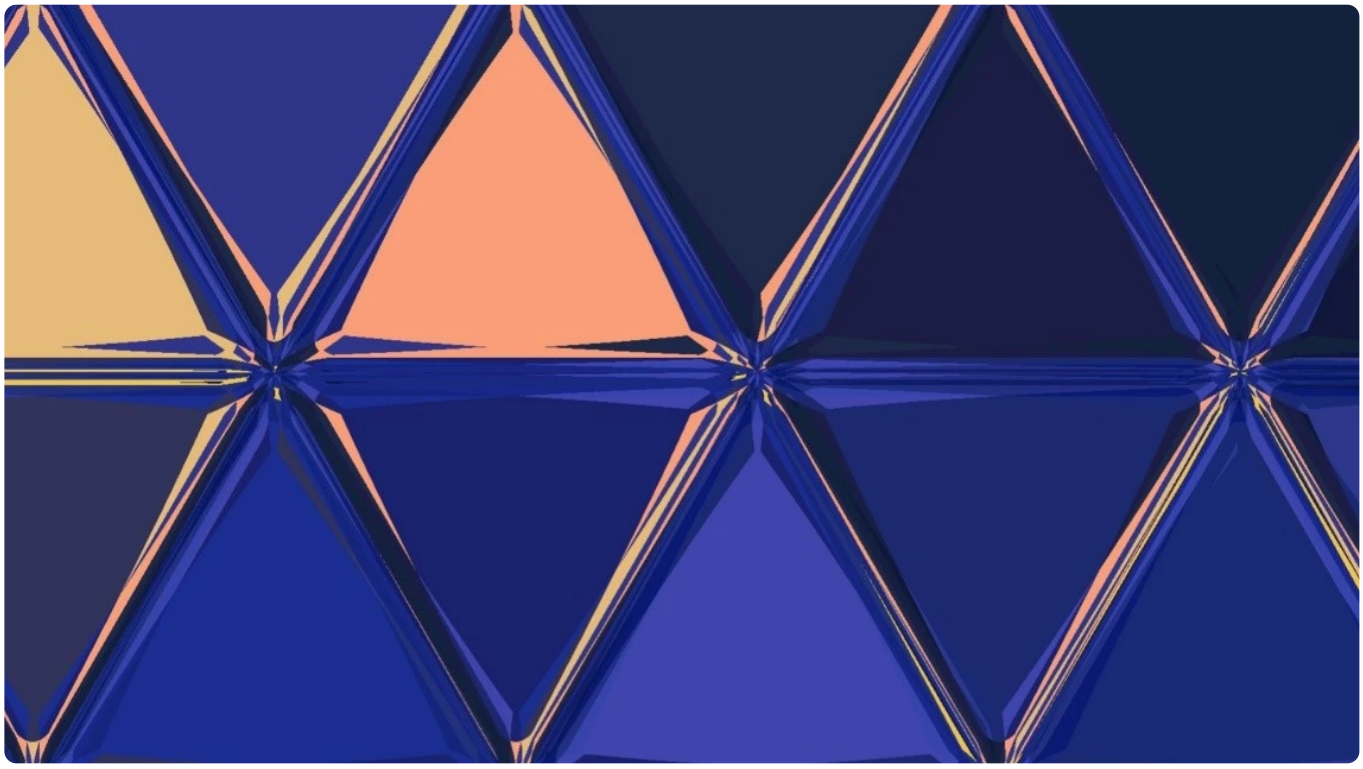
下载APP

复习课（三）| Bigtable

2021-11-10 黄金

《大数据经典论文解读》 课程介绍 >



讲述：王惠

时长 11:05 大小 10.16M

你好，我是黄金。欢迎来到第三期复习课，今天我们来回顾复习下 Bigtable 这篇论文的知识点。

Bigtable 介绍

领资料

在 Bigtable 论文中提到，当年 Google 的很多产品都使用 Bigtable 存储数据，包括索引、谷歌地球、谷歌金融。不管是完成批处理，还是实时数据服务，Bigtable 表现得都很好。也就是说，**Bigtable 不仅擅长顺序读写，也擅长随机读写。**

可运维性强

徐老师在 08 讲中，先是讲了为什么 MySQL 集群难以支撑百万级别的随机读写 IOPS，主要的原因是**可运维性差。**第一，数据分区不灵活，导致随着数据规模的增长，有些分区

数据多，有些分区数据少；第二，服务器扩容不灵活，扩容时要么需要移动大量数据，要么需要成倍增加服务器；第三，故障恢复时只能自动恢复主节点，不能自动恢复备份节点。

那么，Bigtable 是如何解决这些运维问题的呢？

首先，为了应对数据规模的增长，我们需要把数据分配到不同的服务器，这个行为叫做**分区**。Bigtable 的分区方式是为每个分区分配一段连续的**行键**（Row Key），每个分区管理固定大小的数据。当分区数据超过阈值时，比如 128MB，分区就会自动分裂。之所以这样做，是因为数据分布可能是不均匀的，动态分区可以让数据在服务器上分布更均匀。

然后，正是由于每个分区负责的行键区间不固定，就需要一个**组件**帮助客户端在发起请求时，找到行键所在的数据分区。这个组件需要让集群中的所有节点达成共识，确定哪个区间的数据在哪个分区上，由哪台服务器提供服务。它的名字就是 **Chubby**，在后续的论文解读课上我们还会深入学习，目前只需要知道它是**用来在集群中对某件事达成共识**。

另外，Bigtable 中还分离了存储模块和在线服务，采用不同的方法应对扩容和故障恢复。Bigtable 的分区数据叫做 Tablet，存储在 GFS 上，很自然地具备了扩容和容灾的特性。而负责处理读写请求的在线服务是 Tablet Server，Tablet Server 支持动态增加和删除，Bigtable 会把 Tablet 合理地分配给存活的 Tablet Server。

我们应该也知道，**MySQL 的容灾方式是为主服务指定一个备份服务**，主服务挂了就切到备份服务上，但是如果这个备份服务也挂了就麻烦了，我们没有更多的备份服务顶上，所以服务器一旦发生故障，运维人员需要及时处理。

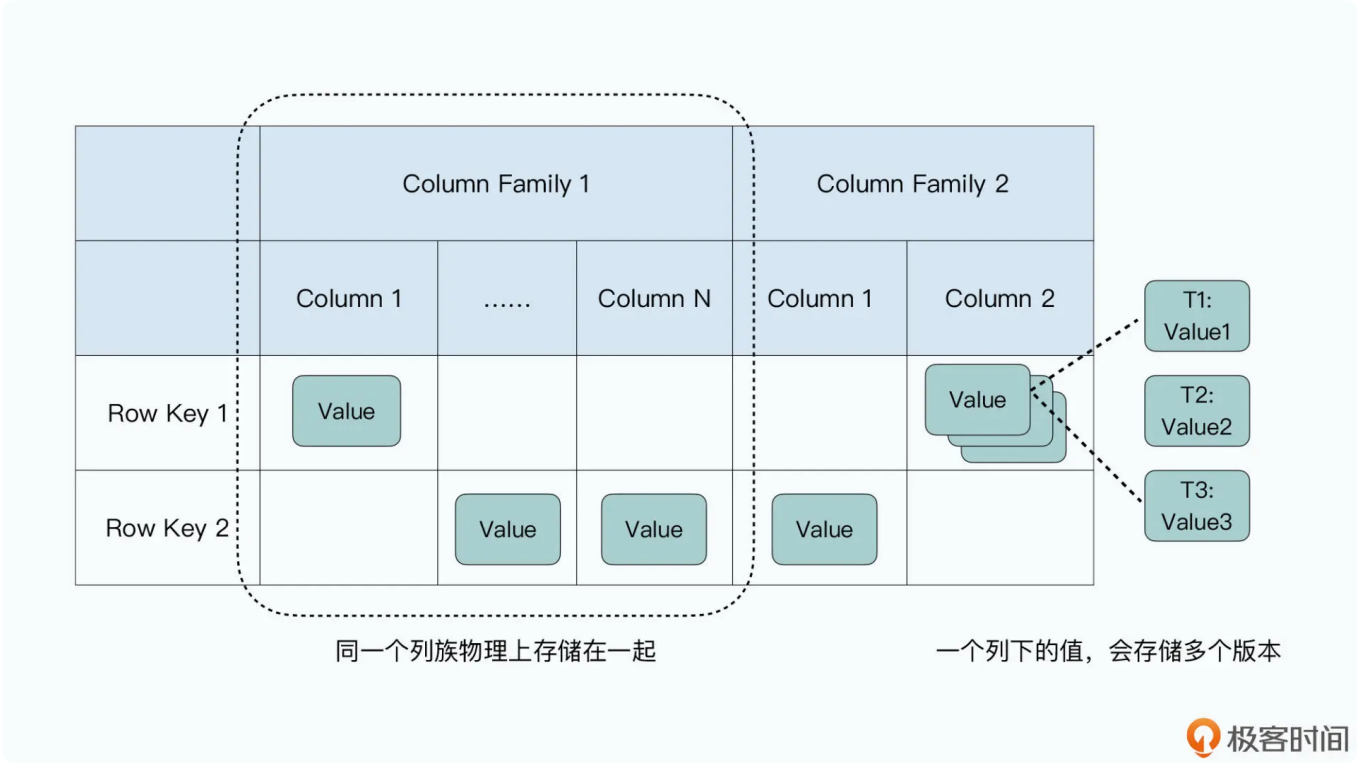
而 **Bigtable 的容灾方式是为主服务指定一个备份服务池**，主服务挂了就从备份服务池中选一个服务作为主服务，如果这个备份服务也挂了，就从池子里再选一个。这种容灾方式，让运维人员可以更从容地处理故障。

数据模型

了解了 Bigtable 的可运维性，接着再让我们一起回顾下 Bigtable 的数据模型。

Bigtable 是一个**有序的稀疏表**。每一条数据有一个行键，通过行键可以原子性地读写一条数据。一条数据包含了多个列族，不同行数据的同一列族内，可以定义不同的列。每一个

列不仅可以保存值，而且可以保存多个版本，每个版本包含了一个时间戳。



Bigtable 的数据是按行键的字段顺序排列的，采用动态行键范围分区，分区被称为 Tablet。刚开始在 Bigtable 里，一张表只有一个分区，随着数据越写越多，分区会一分为二，新分区的边界按**数据密度**划分，以保证两个分区的数据量大致相同。而如果一个分区的数据删除太多，就会和它相邻的分区合并，新的分区负责原来两个分区行键范围的数据。

Bigtable 的列族是把一组相关的列在物理上放在一起存储，不同的列族会通过逻辑组织在一起。当读写操作只涉及某个列族时，Bigtable 只需要处理这个列族对应的物理表即可。**列族不同于列式存储**，因为列族不是按列分开存储的，而是把多个相关的列打包到一起存储，并且允许每一行有不同的具体列。

架构与实现

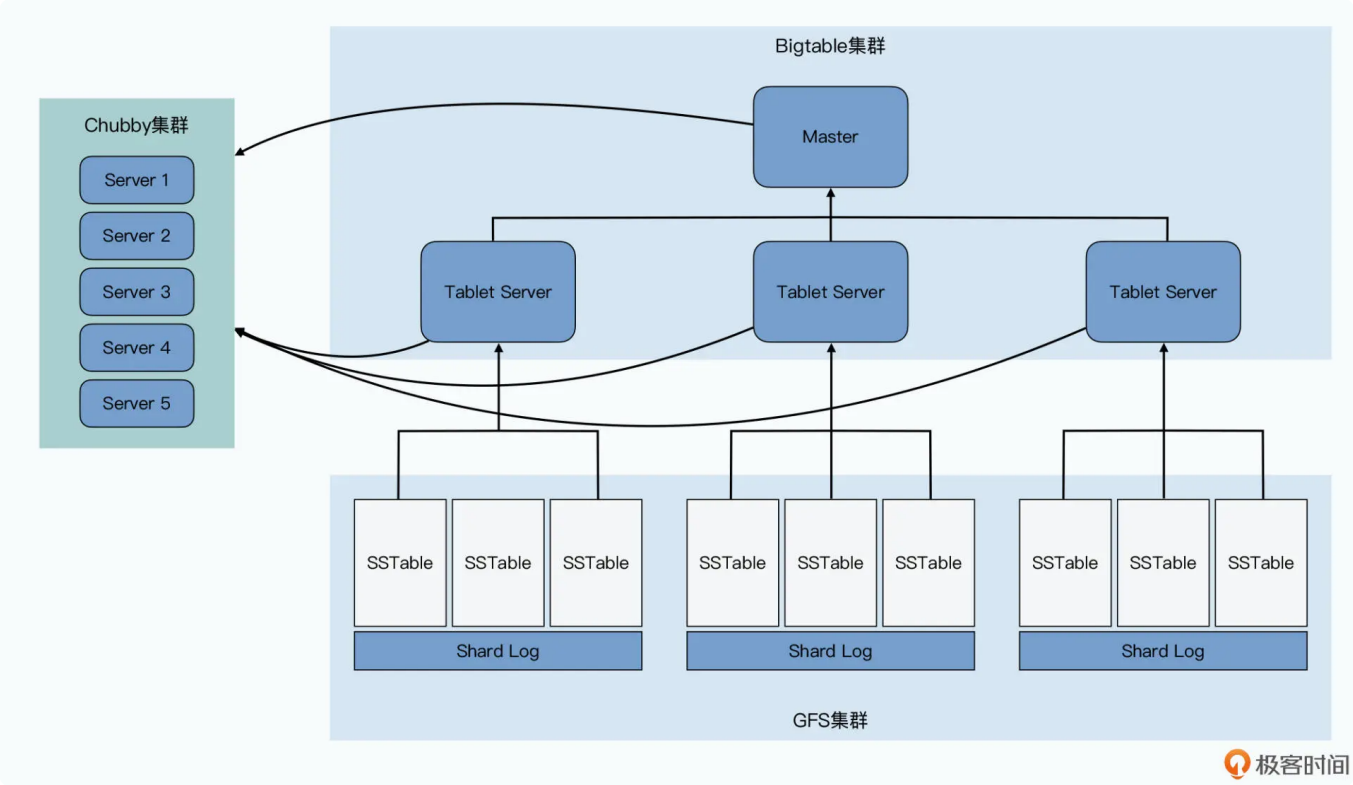
纵观整个 Bigtable 的系统架构，主要是由 4 个组件组成的，它们分别是：

负责提供在线服务的 Tablet Server；

负责存储分区数据 Tablet 的 GFS；

负责调度 Tablet 和调整负载的 Master；

负责分布式协调服务的 Chubby。

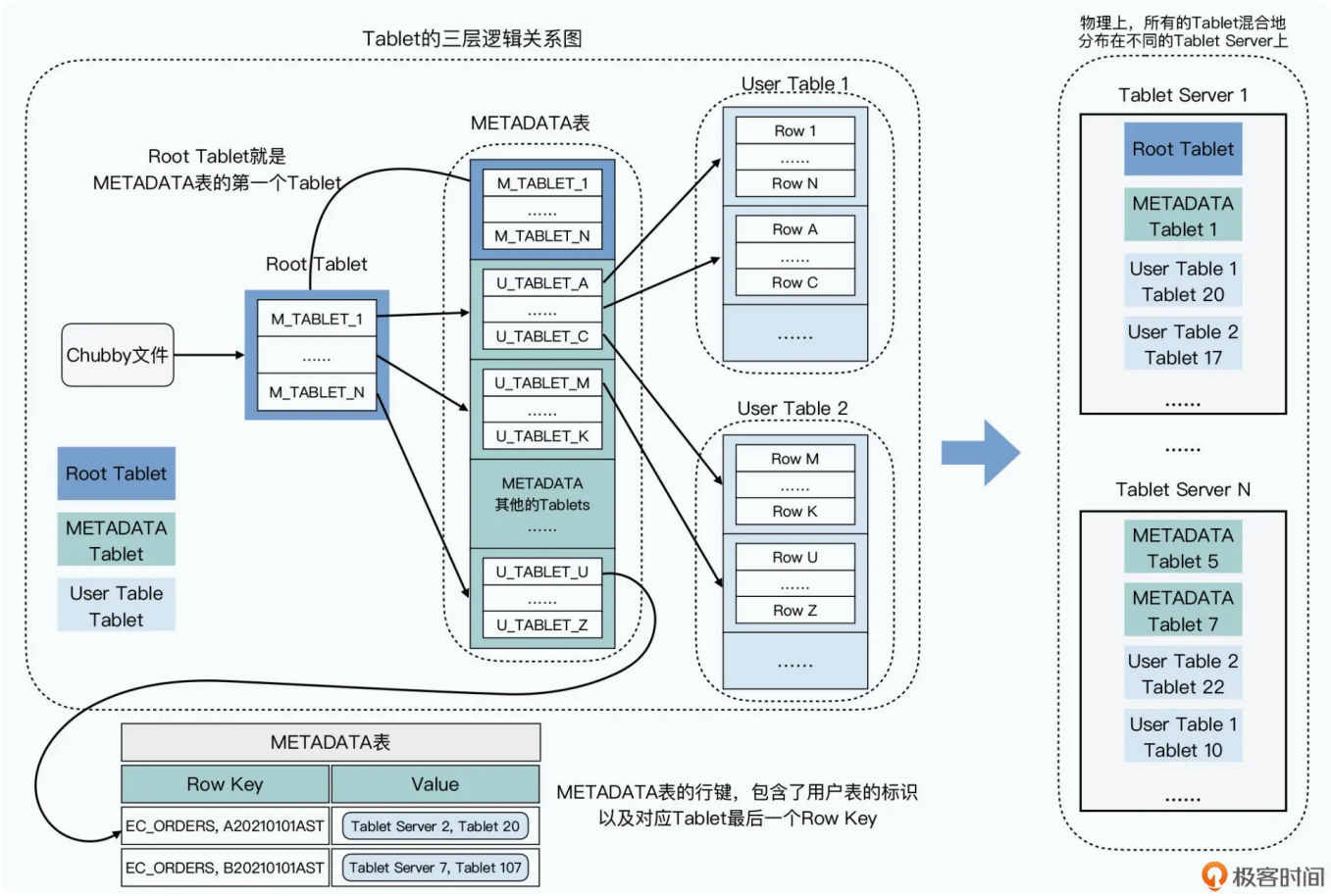


由于采用动态行键范围分区，Bigtable 需要维护行范围到分区的映射关系，以便客户端请求能够路由到正确的 Tablet Server 上。Bigtable 采用了三层树状结构存储数据表的分区信息：

第一层是 Chubby 中的一个文件，它记录了 Root Tablet 所在的地址。

第二层是 Root Tablet，它记录了 METADATA 表的分区信息。METADATA 表用于记录所有数据表的分区信息，而它自身也是一张数据表，包含多个分区。Root Tablet 特别的地方在于它是 METADATA 表的第一个分区，并且永远不会分裂。

第三层是 METADATA 表的其他分区，通过这一层就能找到数据所在的实际分区，以及管理这个分区的 Tablet Server。



Bigtable 会通过 Chubby 追踪 Tablet Server。当新的 Tablet Server 添加到集群中的时候，会向 Chubby 注册自己，Chubby 会通知 Master 有新的 Tablet Server 加入，Master 会把其他 Tablet Server 管理的 Tablet 匀一部分，给新加入的 Tablet Server。

Master 会通过心跳定时检查 Tablet Server 的存活状态，如果 Tablet Server 在 Chubby 的注册信息不在了，则可以认为 Tablet Server 处于失效状态，Master 会把这个 Tablet Server 负责的 Tablet，分配给其他存活的 Tablet Server。

而因为 Bigtable 集群只有一个 Master 服务，如果 Master 挂了，Chubby 就会立即选举一个新的 Master。由于读写操作只和 Chubby、Tablet Server、Tablet 有关，Master 只负责调度 Tablet 和调整负载，所以单 Master 的架构并不会成为 Bigtable 的性能瓶颈。

每秒百万级别的随机读写

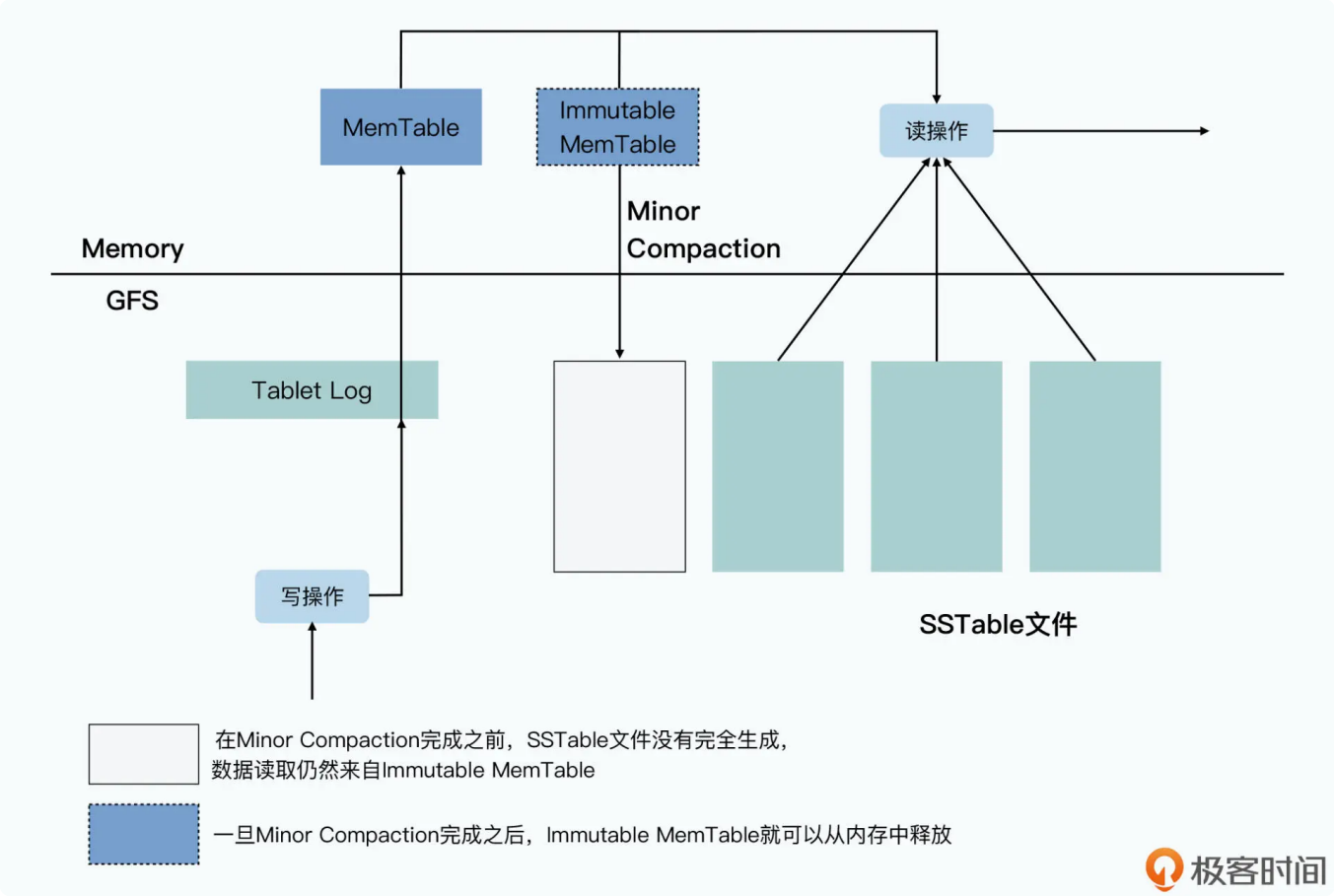
Bigtable 最耀眼的能力在于**可以支撑每秒百万级别的随机读写**。它主要是通过两个方法达成这个目标的：

把随机写变成顺序写；

尽可能从内存中读。

我们先来看看 Bigtable 是如何把随机写变成顺序写的。

当客户端向 Bigtable 发起一次随机写请求，会通过三层树状结构的分区信息，找到对应的分区 Tablet，以及处理写请求的 Tablet Server。Tablet Server 将这次写操作记录到 Commit Log，然后变更内存表 MemTable，当内存表大于某个阈值时，将其作为 SSTable 文件写入磁盘，并清理 Commit Log。



接着我们再看看，Bigtable 是如何通过内存表 MemTable 和 SSTable 文件，完成随机读操作的。

当客户端向 Bigtable 发起一次随机读操作，同样会通过三层树状结构的分区信息，找到对应的分区 Tablet，以及处理读请求的 Tablet Server。Table Server 会先从内存表 MemTable 中查找数据，然后再通过 SSTable 文件的缓存查找数据，如果都找不到的话，它会再加载 SSTable 文件，从中查找。

那么，随着写操作次数的增加，Tablet 下的 SSTable 会越来越多，从而就会影响读操作的性能。因此，Bigtable 会把多个 SSTable 压实成一个 SSTable，只保留对一个行键的最后几次操作，这个行为称为 **Major Compaction**。它不仅能够节约硬盘空间，还能够提升读操作性能。

而除了 Major Compaction 之外，Bigtable 还有一种 Compaction，叫做 **Minor Compaction**，它在内存表大于某个阈值时，会将其作为 SSTable 文件写入磁盘。它的作用是节约内存空间，同时减少内存表恢复时，需要读取的 Commit Log 大小。

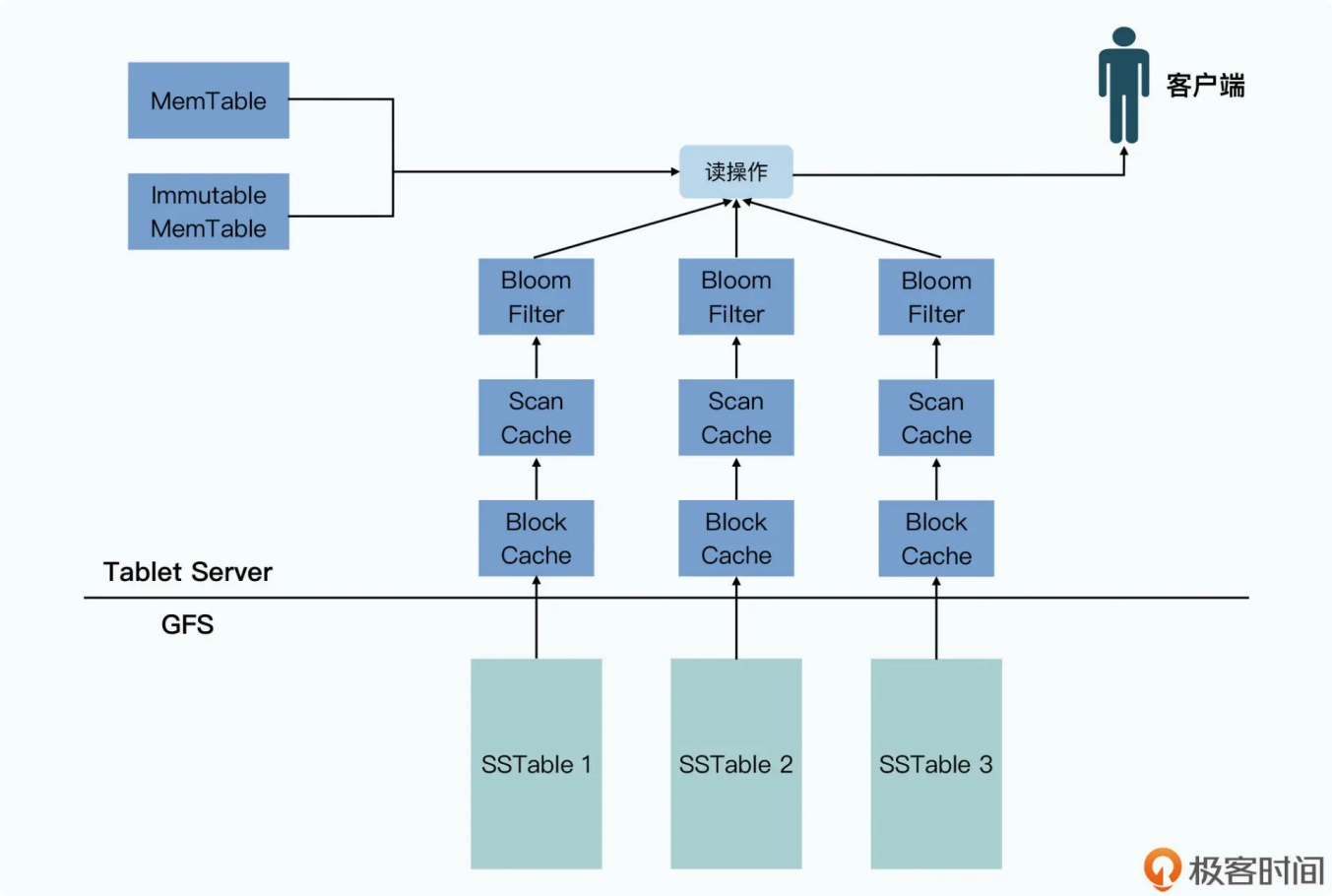
另外，为了提高随机读的性能，Bigtable 其实还做了很多优化。

首先，它会通过一个**布隆过滤器**，确定 SSTable 中是否包含要读取的行键，如果不包含，就快速跳过这个 SSTable；

其次，它会**缓存查过的数据行和邻近的数据行**，以便下次直接从缓存中读取；

再次，**SSTable 文件中包含了索引数据**，这些索引数据紧凑地放在文件尾部，我们通过索引只需要加载 SSTable 的部分数据块，就能确定 SSTable 是否包含要读取的行键，以及行键对应的值；

最后，由于 SSTable 是不变的，**访问 SSTable 不需要加锁**，内存表 MemTable 采用 Copy-On-Write 的方式修改，查询内存表也不需要加锁。



小结

好了，到这里 Bigtable 论文的核心内容我们就复习完了。Bigtable 主要有两个特点，**第一个是具备良好的可运维性**，服务器支持动态伸缩，数据分区支持动态划分，所有的服务出现故障后，在一定程度内都能自动恢复。**第二个是能够支撑每秒百万级别的随机读写**，主要的方法有两个，一是把随机写变成顺序写，二是尽可能从内存中读。

徐老师在 09 和 10 讲都推荐了《数据密集型应用系统设计》这本书，如果你还没有开始阅读的话，我强烈建议你读起来。最近一有空我就会拿起这本书翻一翻，对于理解课程内容和拓展数据系统的知识非常有帮助，你也可以参考看看。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 3 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 复习课（二）| MapReduce

下一篇 复习课（四）| Thrift

精选留言（2）

写留言

Geek_z
2021-11-17

老师后面有没有可能讲下Mesa: Near Real-Time, Scalable Data Warehousing，当下比较流行的Doris之类底层论文

展开

共 1 条评论 >

Dr.森
2021-11-11

优秀的课代表👏👏👏

展开

共 1 条评论 >